Structural advantages for ant colony optimisation inherent in permutation scheduling problems

Montgomery, James; Randall, M; Hendtlass, T

Published in:
Innovations in Applied Artificial Intelligence

DOI:
10.1007/11504894_31

Structural Advantages for Ant Colony Optimisation Inherent in Permutation Scheduling Problems

James Montgomery

No Institute Given

Abstract. When using a constructive search algorithm, solutions to scheduling problems such as the job shop and open shop scheduling problems are typically represented as permutations of the operations to be scheduled. The combination of this representation and the use of a constructive algorithm introduces a bias typically favouring good solutions. When ant colony optimisation is applied to these problems, a number of alternative *pheromone representations* are available, each of which interacts with this underlying bias in different ways. This paper explores both the structural aspects of the problem that introduce this underlying bias and the ways two pheromone representations may either lead towards poorer or better solutions over time. Thus it is a synthesis of a number of recent studies in this area that deal with each of these aspects independently.

Keywords: heuristic search, planning and scheduling.

1 Introduction

Ant Colony Optimisation (ACO) is a constructive metaheuristic that uses an analogue of ant trail pheromones to learn about good features of solutions. ACO belongs to the class of model-based search (MBS) algorithms [1]. In an MBS algorithm, new solutions are generated using a parameterised probabilistic model, the parameters of which are updated using previously generated solutions so as to direct the search towards promising areas of the solution space. The model used in ACO is known as *pheromone*, an artificial analogue of the chemical used by real ants to mark trails from the nest to food sources. While pheromone used by real ants is deposited on the ground they traverse, artificial pheromone can often be associated with a variety of features that characterise and distinguish solutions. Choosing which features to associate pheromone with is an important design decision when adapting ACO to suit a particular problem. Indeed, recent work by Blum and Sampels [2] and Blum and Dorigo [3] has revealed that the choice of pheromone representation can introduce a distinct and potentially unhelpful bias to an ACO search.

This paper considers how the structure of a number of scheduling problems can actually assist the performance of ACO, especially if a particular pheromone

representation is used. Previous work by Montgomery, Randall and Hendtlass [4] examines the structure of the space in which ants build solutions. In contrast, Blum and Sampels [2] and Blum and Dorigo [3] study the frequency with which individual pheromone values are updated given different pheromone representations. This paper is a synthesis of both approaches to understanding bias in ACO. The well-known job-shop and open-shop scheduling problems (JSP and OSP respectively) are used both to illustrate these biases and to highlight the interesting structure these problems exhibit when solved by ACO.[1] Understanding the mechanisms of these biases establishes that they are enduring features of these kinds of scheduling problems, which allows for the consistent and effective application of optimisation techniques such as ACO.

Section 2 describes the JSP and OSP and the way in which solutions to these problems are produced by ACO and other constructive algorithms. Section 3 describes the structural aspects of these problems that favour good solutions, while Section 4 considers the way different pheromone representations react to this structure and lead to the reinforcement of either poorer or better solutions. Section 5 summarises the findings.

2 ACO Applied to Shop Scheduling Problems

The JSP and OSP are well-known scheduling problems with applications in manufacturing [6]. An instance of either problem consists of a set of *operations* $\mathcal{O} = \{o_1, o_2, \ldots, o_{|\mathcal{O}|}\}$ partitioned into the *jobs* to which they belong $\mathcal{J} = \{J_1, J_2, \ldots, J_{|\mathcal{J}|}\}$ and the *machines* $\mathcal{M} = \{M_1, M_2, \ldots, M_{|\mathcal{M}|}\}$ on which they must be processed. In both problems, only one operation from a job may be processed at any given time, only one operation may use a machine at any given time and operations may not be pre-empted. In the JSP, precedence constraints impose a total ordering on the operations within each job (i.e., there is a fixed sequence in which operations must be processed), while operations may be processed in any order in the OSP. Each operation o_i has a non-negative processing time $p(o_i)$, and the aim of both problems is to minimise the total amount of time to complete all jobs, called the *makespan*. The makespan of a solution s is denoted by $C(s)$. Blum and Sampels [2] describe a generalisation of these problems where operations within each job are also partitioned into *groups*, with precedence constraints applying within groups. This generalisation is called the group shop scheduling problem (GSP). In the JSP, each operation is assigned its own group (i.e., precedence constraints apply between operations), while in the OSP all operations within a job belong to a single group (i.e., there are no existing precedence constraints between operations). Given an existing JSP or OSP instance and adjusting the number, and hence size, of groups, a range of problem instances may be constructed with characteristics intermediate between the JSP and OSP.

[1] The JSP and OSP are also the subject of the work by Blum and Sampels [5] and Blum and Dorigo [3], which allows for concurrent validation of results presented in this paper.

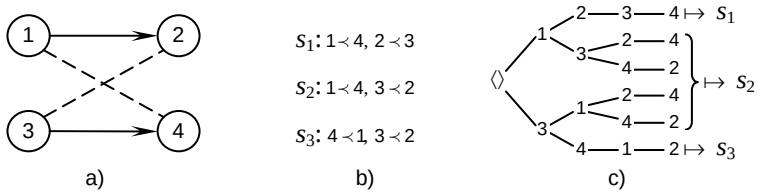

Fig. 1. A JSP instance described by Blum and Sampels [2]. a) A small JSP instance with $\mathcal{O} = \{1, 2, 3, 4\}$, $\mathcal{J} = \{J_1 = \{1, 2\}, J_2 = \{3, 4\}\}$, $1 \prec 2$, $3 \prec 4$, $\mathcal{M} = \{M_1 = \{1, 4\}, M_2 = \{2, 3\}\}$, $p(1) = p(4) = 10$, $p(2) = p(3) = 20$. $i \prec j$ indicates i must be processed before j. b) The three solutions to this problem described in terms of the relative order of operations that require the same machine. $C(s_1) = C(s_3) = 60$, $C(s_2) = 40$. c) The construction tree for this problem showing the six sequences that may be produced and the solutions to which they correspond

It is common to represent instances of these problems as disjunctive graphs, where directed arcs indicate existing precedence constraints (as exist in the JSP for instance) and undirected arcs exist between operations that either require the same machine or are part of the same job but have no pre-existing precedence constraints between them. Operations connected by undirected arcs can be referred to as being *related* [5]. Fig. 1 shows the disjunctive graph representation of a small JSP instance consisting of two jobs, both of two operations each. A schedule for such problems may be created by assigning directions to undirected arcs in the disjunctive graph to create a directed acyclic graph. Each operation is then scheduled as early as possible given the precedence constraints imposed by this directed graph. The *list scheduler algorithm* is a constructive algorithm for these problems that ensures that cycles cannot be created in the disjunctive graph. The algorithm creates a permutation of the operations to be scheduled by successively choosing from those operations whose required predecessors have already been placed in the permutation. The relative order of related operations is determined by their relative positions in the permutation.

In ACO, solutions are built as sequences of *solution components*, which corresponds quite naturally with the list scheduler algorithm, provided that operations are used as solution components. In this paper a sequence of solution components is denoted by $\mathfrak{s}$, while the solution represented by the sequence is denoted by $X(\mathfrak{s})$ or s. The set of sequences that represent a solution s is denoted by $\mathfrak{S}(s)$.

3 Bias Inherent in Constructive Algorithms

At each step of a constructive algorithm a decision is made concerning which solution component to add to the sequence of solution components already built. The set of available solution components is determined by problem constraints and typically excludes those components already included in the partial sequence. Thus constructive algorithms implicitly explore a tree of constructive

decisions, or construction tree, where the root corresponds to the empty sequence $\langle\rangle$ and leaves correspond to complete sequences and hence, to solutions. We denote a construction tree by $\mathcal{T}$.

The topology of the construction tree is defined by the nature of the problem being solved and the solution components used. The constructive algorithm also defines the mapping from sequences to solutions. When applying ACO to the GSP, the mapping from sequences to solutions is typically not uniform. Consider the JSP depicted in Fig. 1. There are three distinct solutions, yet six feasible sequences representing those solutions. Of these, four correspond to solution s_2, thereby introducing a *representation bias* [4] in favour of solution s_2.

Definition 1. *A constructive algorithm applied to a combinatorial optimisation problem is said to have a* representation bias *if there exist two solutions s_1 and s_2 such that $|\mathfrak{S}(s_1)| \neq |\mathfrak{S}(s_2)|$.*

The remainder of this section considers the use of a list scheduler algorithm which selects each solution component probabilistically using a uniform random distribution over the available components at each step. This algorithm is hereafter referred to as $\text{ACO}_{\text{undir}}$ (i.e., undirected ACO). Using such an algorithm, the probability of choosing a particular component at a given node in a construction tree is inversely proportional to the number of alternative components at that node. Consequently, sequences found on paths with fewer alternatives at each node are more likely to be discovered than those on paths with more alternatives at each node. In the example JSP, the probability of each of the sequences corresponding to solutions s_1 and s_3 is twice that for any of the four sequences corresponding to solution s_2, so that overall $P(s_1) = P(s_3) = 0.25$ while $P(s_2) = 0.5$. This constitutes a *construction bias* [4].

Definition 2. *A construction tree $\mathcal{T}$ has a* construction bias *if there exist two nodes in T such that their heights are equal yet their degrees are not equal.*

In problems where every sequence of solution components represents a feasible solution, the degree of nodes in the construction tree is uniform within each level. Such problems consequently do not have a construction bias. GSP instances with at least two groups for one of the jobs all have a construction bias, while the OSP (i.e., a GSP instance with one group per job) does not, as all permutations of operations are permissible.

Construction trees for the GSP have an interesting structure which places these two biases against each other, each in favour of one of two different kinds of solution.

In an investigation of the poor performance of ACO applied to the GSP when using certain pheromone representations, Blum and Sampels [2] found that sequences corresponding to poor solutions tend to have runs of operations from the same job. They measure this characteristic of sequences by introducing a *line scheduling factor*,[2] given by $f_{ls}(\mathfrak{s}) = \left(\sum_{i=1}^{|\mathcal{O}|-1} \delta(\mathfrak{s}, i)\right) / \left(|\mathcal{O}| - |\mathcal{J}|\right)$ where

[2] Blum [7] also refers to this measure simply as a *sequencing* factor, denoted by f_{seq}.

$\mathfrak{s}[i]$ is the operation in the i^{th} position of $\mathfrak{s}$, and $\delta(\mathfrak{s}, i) = 1$ if $\mathfrak{s}[i]$ belongs to the same job as $\mathfrak{s}[i+1]$, 0 otherwise. Hence, the value of f_{ls} is in $[0, 1]$, where 1 indicates that all operations for each job are contiguous, while 0 indicates that no pairs of operations from the same job are adjacent in the sequence.

Sequences with a high line scheduling factor generally correspond to poor solutions to these problems. Intuitively this is to be expected as good schedules allow operations from different jobs to run in parallel. A sequence in which all operations from one job appear in a contiguous group can produce a schedule which contains lengthy delays for other jobs' operations, which must wait for operations from the first job to finish. This intuitive claim is born out by empirical results. The top row of Fig. 2 plots the mean f_{ls} value of sequences for each solution against the cost of the solution represented for a nine operation, three job, three machine JSP and OSP (both with a similar structure to the JSP depicted in Fig. 1).

In GSP instances that are not OSP instances, a construction bias always exists in favour of solutions with a high line scheduling factor. This is most evident in the JSP. In a JSP with n jobs, n operations are available to be added to the sequence at each step (i.e., one from each job) until all the operations from one of the jobs have been added to the sequence, after which $n-1$ operations are available. As each job's set of unscheduled operations becomes empty, the number of available operations becomes smaller. Thus, selecting an operation from the same job as that last added to the sequence decreases the number of steps until that job's set of unscheduled operations becomes empty, and consequently makes it more likely that the same will have to be done with operations from other jobs later in solution construction. Consider a JSP with n jobs of m operations each. A sequence with $f_{ls} = 1$ can be produced on a path with m steps of n options, followed by m steps of $n-1$ options, m steps of $n-2$ options and so on, finishing with m steps of 1 option only. Denote this sequence by $\mathfrak{s}^{f_{ls}=1}$. Consider an alternative sequence constructed by selecting an operation from each job in a round-robin fashion, which accordingly has $f_{ls} = 0$. The path for such a sequence will have $(m-1) \cdot n + 1$ steps at which every job has at least one remaining operation to be scheduled, followed by $n-1$ steps with decreasing numbers of options, $n-1$, $n-2$, ..., 1, as each job's set of unscheduled operations becomes empty. Denote this sequence by $\mathfrak{s}^{f_{ls}=0}$.

The probability of a sequence being produced by $\text{ACO}_{\text{undir}}$ is the inverse of the product of the number of options at each step. Accordingly, $P(\mathfrak{s}^{f_{ls}=1}) = \left(\prod_{i=0}^{n-1}(n-i)^m\right)^{-1}$, while $P(\mathfrak{s}^{f_{ls}=0}) = \left(n^{(m-1)\cdot n+1} \cdot (n-1)!\right)^{-1}$. In general, $P(\mathfrak{s}^{f_{ls}=1}) > P(\mathfrak{s}^{f_{ls}=0}) \quad \forall\, m, n > 1$.

The disparity in probability between sequences with $f_{ls} = 1$ and those with $f_{ls} = 0$ is greatest on the JSP, and diminishes as operation precedence constraints are eased (i.e., in GSP instances with groups containing increasing numbers of operations), becoming zero in OSP instances. Thus sequences corresponding to poor solutions, which typically have a high line scheduling factor, are likely to have a relatively high probability of being found in the construction trees for

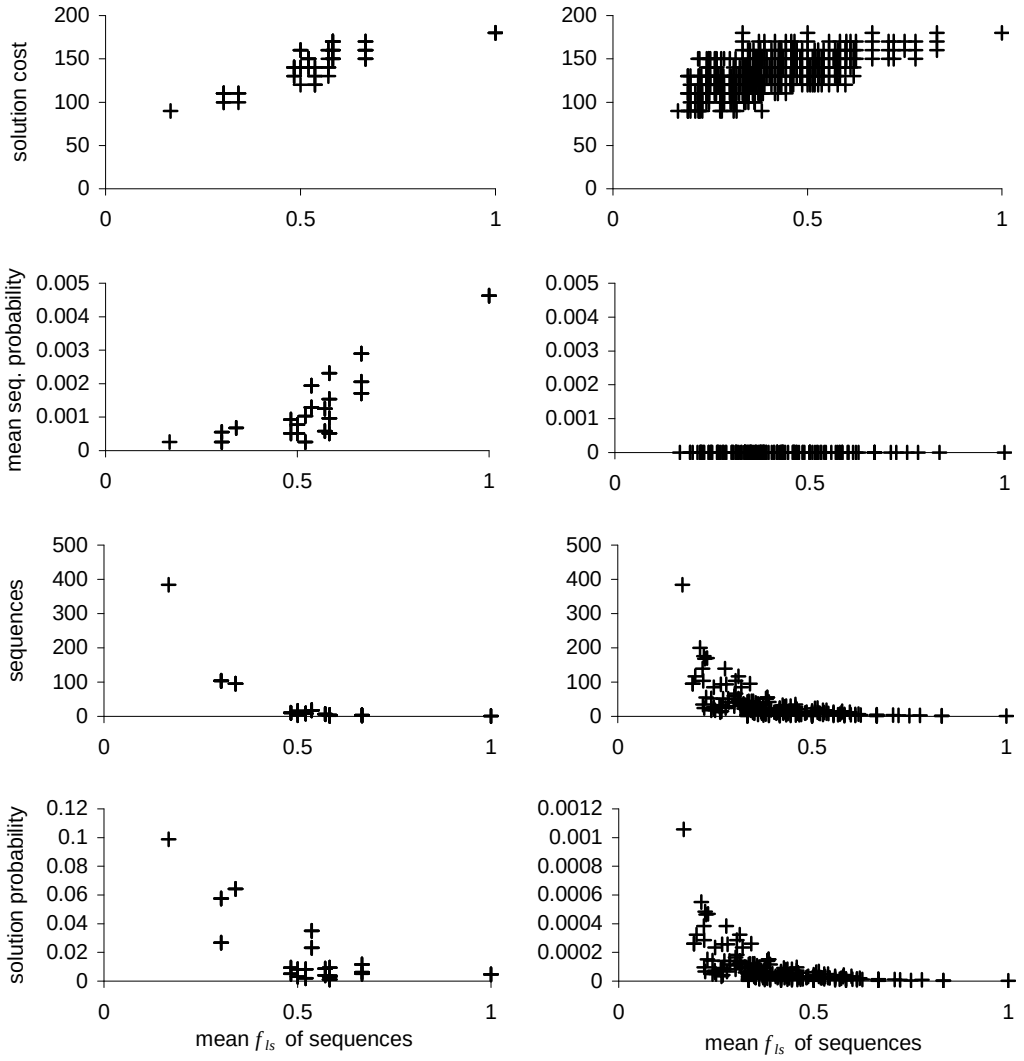

Fig. 2. Mean f_{ls} values of solutions' sequences against: solution cost (top row); mean probability of solutions' sequences (second row); number of sequences per solution (third row); and solution probability (bottom row) for a nine operation, three job, three machine JSP (left) and OSP (right)

JSP and GSP instances (excluding OSP instances). This is illustrated in the second row of Fig. 2.

However, solutions represented by sequences with predominantly high line scheduling factors are generally represented by fewer sequences, across all GSP instances. The third row of Fig. 2 plots the mean line scheduling factor of solutions' sequences against the number of sequences representing that solution.

Intuitively, sequences with a high f_{ls} value can tolerate only small perturbations before the solution represented changes. Certainly, in the JSP, a sequence with $f_{ls} = 1$ can only be altered slightly before the relative order of related operations is changed and the sequence represents a different solution. Accordingly, the lower the line scheduling factor, the easier it is to perturb the sequence without changing the relative order of related operations. This suggests that low cost solutions, which are generally represented by sequences with a low f_{ls} value, are overrepresented in the construction tree.

Indeed, the representation bias, which typically favours good solutions to these problems, can overwhelm the construction bias that typically favours poorer solutions. The fourth row of Fig. 2 plots the mean line scheduling factor of solutions' sequences against the overall probability of finding that solution using $\text{ACO}_{\text{undir}}$.

In moderate to large problem instances it becomes impossible to perform a complete exploration of the construction tree and hence to analyse the impact of construction and representation biases. While these biases must still be present, for the reasons given above, any search algorithm can at best produce a sample of the many feasible solutions to such instances. However, although the effects of these biases cannot be observed on larger instances, the mechanisms that drive them do have an impact on the different pheromone representations that an ACO algorithm may use.

4 Pheromone and Construction Biases

Constructive decisions in ACO are biased by pheromone information, which represents the learned utility of adding a particular solution component given the current state of the sequence and/or solution under construction.[3] A pheromone representation is a collection of pheromone values that individually correspond to some characteristic of either a sequence or the solution it represents. *Solution characteristics* may either correspond to the solution components used to build a solution or to some aggregate feature of a solution induced by a number of solution components [8]. Pheromone values for each solution characteristic are increased in proportion to the quality of the solutions with those characteristics produced at each iteration of the algorithm. The relative value of pheromone associated with each solution characteristic influences the selection of solution components in later iterations.

Two pheromone representations for the GSP are considered in this paper. $\mathsf{PH}_{\mathsf{suc}}$, used in early ACO algorithms for these problems, associates a pheromone value with pairs of operations that may be placed in succession (including an artificial start node that is not part of the original problem description). Hence the solution characteristic (o_1, o_2) from $\mathsf{PH}_{\mathsf{suc}}$ relates to the learned utility of

[3] Constructive decisions in ACO are also typically biased by a problem-specific heuristic measure of the utility of adding a component, but this is not considered here in order to simplify the analyses performed.

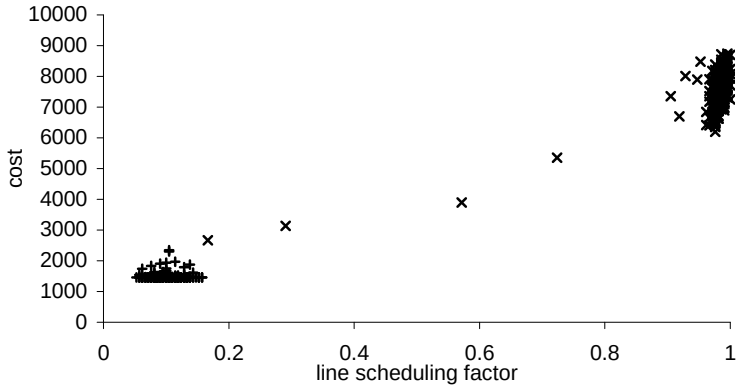

Fig. 3. f_{ls} values of samples of 300 sequences produced by ACO with $\mathsf{PH}_{\mathsf{suc}}$ (shown as $\times$) and $\mathsf{PH}_{\mathsf{rel}}$ (shown as $+$) against cost of solutions represented. All points for $\mathsf{PH}_{\mathsf{rel}}$ have $f_{ls} \in (0.05, 0.16)$

placing operation o_2 immediately after operation o_1 in a sequence. $\mathsf{PH}_{\mathsf{rel}}$, a recently developed pheromone representation introduced by Blum and Sampels [5], associates a pheromone value with pairs of *related* operations to learn which operation should precede the other. Hence the solution characteristic (o_1, o_2) from $\mathsf{PH}_{\mathsf{rel}}$ relates to the learned utility of scheduling o_1 before o_2, i.e., at any location in the sequence before o_2. When considering a candidate operation o_1, $\mathsf{PH}_{\mathsf{rel}}$ makes use of a number of pheromone values, as a candidate operation may be related to many as yet unscheduled operations. Blum and Sampels [5] take the minimum pheromone value associated with these characteristics.

In empirical work conducted by Blum and Sampels [2], and in the current investigation, $\mathsf{PH}_{\mathsf{suc}}$ was found to perform poorly on the GSP. Its performance is worst on the JSP, but improves as problem constraints are eased such that its performance is very good on the OSP. Blum and Sampels observed high f_{ls} values (up to 1) for sequences produced by $\mathsf{PH}_{\mathsf{suc}}$ applied to GSP instances other than the OSP. In contrast, f_{ls} values when using $\mathsf{PH}_{\mathsf{rel}}$ were consistently low (less than 0.1) across the JSP, GSP and OSP. This result has been found across a range of instances of varying size. As was found by Blum and Sampels, and illustrated in Section 3, sequences with a high f_{ls} value typically represent poor solutions to these problems, a result which holds regardless of problem size. Fig. 3 plots f_{ls} values against solution cost for sequences produced by ACO algorithms using $\mathsf{PH}_{\mathsf{suc}}$ and $\mathsf{PH}_{\mathsf{rel}}$ applied to the `la38` JSP instance.[4] Data were collected by sampling every 100^{th} sequence produced by an ACO algorithm producing a total of 30,000 sequences.[5]

[4] This instance is part of a benchmark JSP set described by Lawrence [9].

[5] The actual algorithm used is a modification of Ant Colony System from which heuristic information and its greedy bias (q_0) have been removed.

An insight into the strong bias $\mathsf{PH}_{\mathsf{suc}}$ exhibits towards solutions with a high f_{ls} value can be obtained in a number of ways. Blum [7] introduces the concept of a *competition balanced system*, which in terms of ACO is defined as a pheromone representation consisting of solution characteristics that appear in the same number of sequences produced by the algorithm. If a pheromone model applied to a particular problem instance is not a competition-balanced system, Blum states that bias may be observed. Certainly, when using $\mathsf{PH}_{\mathsf{suc}}$ with constrained GSP instances (such as the JSP), solution characteristics corresponding to placing two operations from the same job in succession appear in proportionally more sequences than those for which it is not the case. In contrast, solution characteristics from $\mathsf{PH}_{\mathsf{rel}}$ that are associated more strongly with sequences with a low f_{ls} value appear in a greater number of sequences than those characteristics that are not. Thus, in problems where a high f_{ls} value is strongly predictive of a high solution cost, use of $\mathsf{PH}_{\mathsf{suc}}$ will make good solutions increase the pheromone associated with poor solutions, whereas use of $\mathsf{PH}_{\mathsf{rel}}$ will result in even poor solutions increasing pheromone associated most strongly with good solutions.

Consideration of the structure of these problems, described in Section 3, reveals why the solution characteristics from these two pheromones are so strongly biased towards different kinds of sequences and hence, solutions. Given that selecting an operation from the same job as that most recently selected decreases the likelihood that successive pairs of operations placed later will be selected from different jobs, those solution characteristics from $\mathsf{PH}_{\mathsf{suc}}$ that correspond to placing successive operations from different jobs are also less likely to appear in those sequences. In contrast, partially constructed sequences with a low f_{ls} value restrict the set of available operations less, and so still allow successive operations from the same job to be placed. Thus, the same mechanism that introduces a construction bias (which has little detectable effect on larger instances) does have an effect on the distribution of solution characteristics from $\mathsf{PH}_{\mathsf{suc}}$ in the construction tree. Conversely, many of the operation precedence relationships established by sequences with a high f_{ls} value are largely restricted to those sequences, and are not present in those sequences that may be perturbed while maintaining the solution represented. Sequences with a high f_{ls} value will still contain some of those operation precedence relationships that appear in better solutions, and so overall the number of sequences that these precedence relationships appear in is relatively high. The representation bias in these problems serves to accentuate the effect, as all sequences for a single solution exhibit the same solution characteristics in $\mathsf{PH}_{\mathsf{rel}}$.

5 Conclusions

The structure of the GSP, which includes the well-known JSP and OSP, serves to bias constructive searches towards good solutions. However, on medium to large instances the relative difference between competing solutions becomes negligible given the comparatively large number of solutions overall. Nevertheless, the presence of underlying biases in the construction trees for these problems

produces a bias in the various pheromone representations that may be used by ACO. Associating pheromone with pairs of successive operations in a sequence ($\mathsf{PH}_{\mathsf{suc}}$) performs poorly because the construction path for those sequences that represent poor solutions necessarily restricts alternatives, thereby increasing the number of sequences in which solution characteristics of poor solutions appear. Conversely, learning the relative order of related operations ($\mathsf{PH}_{\mathsf{rel}}$) performs well because in that pheromone representation characteristics of poor solutions can only appear in a small number of sequences as small perturbations to those sequences change these characteristics. Understanding the mechanisms underlying these different behaviours of ACO applied to these problems establishes that they are enduring features, and so supports the effective application of ACO to these problems. The interesting and advantageous structure of these problems suggests the possible existence of other problems that have a structure that may be similarly exploited by the use of a carefully chosen pheromone representation to increase the probability of finding good solutions. It also suggests that there may be problems whose structure cannot be exploited and which require additional heuristic techniques to counter any inherent unfavourable biases.

References

1. Zlochin, M., Dorigo, M.: Model-based search for combinatorial optimization: A comparitive study. In: 7th International Conference on Parallel Problem Solving from Nature (PPSN 2002). (2002) 651–662
2. Blum, C., Sampels, M.: When model bias is stronger than selection pressure. In Guervós, J.M., et al., eds.: 7th International Conference on Parallel Problem Solving from Nature (PPSN2002). Volume 2439 of Lecture Notes in Computer Science., Springer-Verlag (2002) 893–902
3. Blum, C., Dorigo, M.: Deception in ant colony optimisation. In Dorigo, M., et al., eds.: 4th International Workshop on Ant Colony Optimization and Swarm Intelligence, ANTS 2004. Volume 3172 of Lecture Notes in Computer Science., Springer-Verlag (2004) 118–129
4. Montgomery, J., Randall, M., Hendtlass, T.: Search bias in constructive metaheuristics and implications for ant colony optimisation. In Dorigo, M., et al., eds.: 4th International Workshop on Ant Colony Optimization and Swarm Intelligence, ANTS 2004. Volume 3172 of Lecture Notes in Computer Science., Springer-Verlag (2004) 390–397
5. Blum, C., Sampels, M.: Ant colony optimization for FOP shop scheduling: A case study on different pheromone representations. In: 2002 Congress on Evolutionary Computation. (2002) 1558–1563
6. Blum, C., Sampels, M.: An ant colony optimization algorithm for shop scheduling problems. Journal of Mathematical Modelling and Algorithms **3** (2004) 285–308
7. Blum, C.: Theoretical and practical aspects of ant colony optimization. PhD thesis, Université Libre de Bruxelles, Belgium (2004)
8. Montgomery, J., Randall, M., Hendtlass, T.: Automated selection of appropriate pheromone representations in ant colony optimisation. Artificial Life (to appear)
9. Lawrence, S.: Resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques (supplement). Technical Report, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh (1984)